-

A Sierra Monitor Company

Driver Manual
(Supplement to the FieldServer Instruction Manual)

FS-8700-132 Gamewell FCI E3 Series Serial Driver

APPLICABILITY & EFFECTIVITY

Effective for all systems manufactured after September 2008

Driver Version:	**1.01**
Document Revision:	**4**

TABLE OF CONTENTS

1. Gamewell FCI-E3 Series Serial Driver Description

The Gamewell FCI-E3 Series System Control Units are manufactured by Fire Control Instruments. An E3 Panel with an enabled serial port can transmit data to a FieldServer which can, in turn, make the data available to other devices including those which communicate using different protocols (e.g. BACnet)

This passive Client driver does not poll for data, nor does it send data or commands to the E3. Messages received from the E3 are analyzed and are then either discarded or used to update the FieldServer's internal Data Arrays. The action depends on which types of events, the type of device generating the event and device address the FieldServer has been configured to process. Once stored in the FieldServer the data is available to be read or written using other protocols such as BACnet.

No automatic panel data synchronization technique exists. The data in the FieldServer and the panel status have to be synchronized manually. This typically requires a panel reset.

Since the driver cannot send data or commands to the E3 it cannot be used to acknowledge, silence or reset alarms and other events.

The driver can process messages from networked panels. The driver connects to the main panel. Subsidiary panels are configured to send event data to the main panel which then sends messages to the FieldServer. Node information is sent in the line preceding the event and the driver uses this to determine the panel at which the event originated and to store data appropriately.

The driver provides both Client and Server emulation. The Server side of the driver is intended to support FieldServer's Quality Assurance program and is not intended to provide complete emulation of an E3 and is thus not fully documented. Should you require the Server side functionality to be documented and enhanced, please contact FieldServer's sales group.

Max Nodes Supported

FieldServer Mode	Nodes	Comments
Client	1	1 Node per serial port. If there is more than one alarm panel they can be networked and configured to send event data to the primary panel. The driver can process messages which identify the node of origin.
Server	N/A	Server side is not supported. See description.

2. Driver Scope of Supply

2.1. Supplied by FieldServer Technologies for this driver

FieldServer Technologies PART #	Description
FS-8917-16	Pigtail cable for RJ45 Port for RS-232 use
FS-8700-132	Driver Manual.

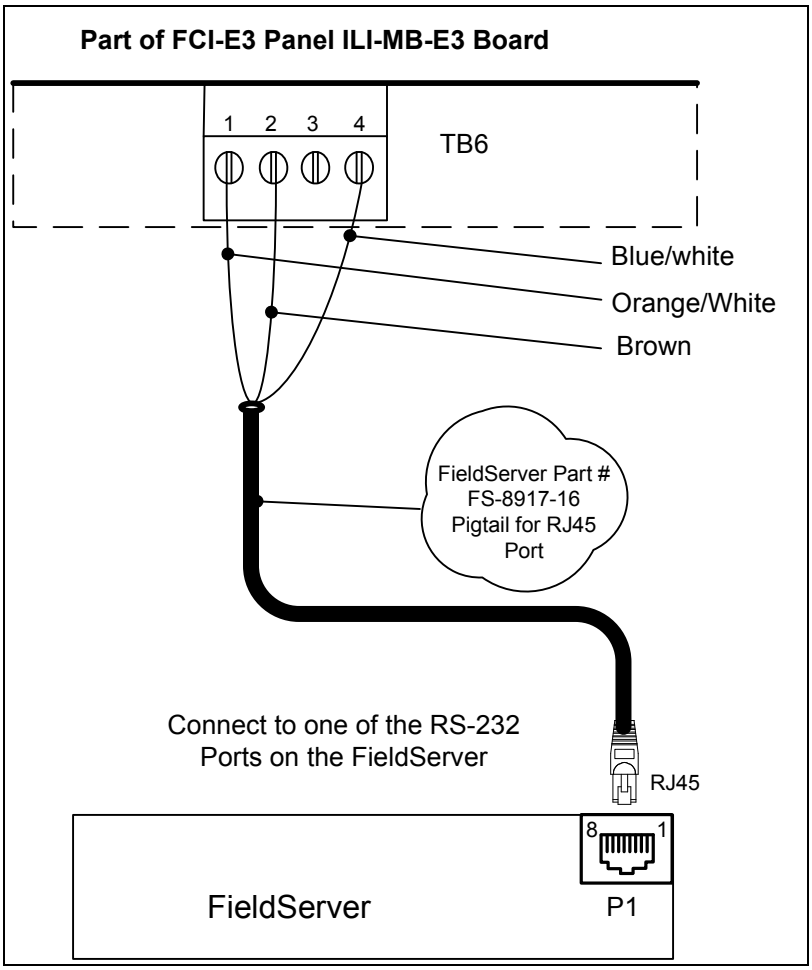
2.2. Provided by the Supplier of 3rd Party Equipment

2.2.1. Required 3rd Party Hardware

ILI-MB-E3 Board

3. Hardware Connections

The FieldServer is connected to the FCI-E3 panel as shown in the connection drawing.



Connector Pinouts

Wire Color	RJ-45		FCI-E3	
	Pin	Signal	Signal	Pin
Brown	1	Rx	+ve	2
White/Orange	8	Tx	-ve	1
Blue/white	5	GND	Signal Common Reference	4

4. Configuring the FieldServer as a Gamewell FCI-E3 Series Serial Driver Client

For a detailed discussion on FieldServer configuration, please refer to the FieldServer Configuration Manual. The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the FieldServer (See ".csv" sample files provided with the FieldServer).

This section documents and describes the parameters necessary for configuring the FieldServer to communicate with a FCI Series E3.

4.1. Data Arrays/Descriptors

The configuration file tells the FieldServer about its interfaces, and the routing of data required. In order to enable the FieldServer for Gamewell FCI-E3 Series Serial Driver communications, the driver independent FieldServer buffers need to be declared in the "Data Arrays" section, the destination device addresses need to be declared in the "Client Side Nodes" section, and the data required from the Servers needs to be mapped in the "Client Side Map Descriptors" section. Details on how to do this can be found below.

Note that in the tables, * indicates an optional parameter, with the bold legal value being the default.

Section Title		
Data_Arrays		
Column Title	**Function**	**Legal Values**
Data_Array_Name	Provide name for Data Array	Up to 15 alphanumeric characters
Data_Array_Format	Provide data format. Each Data Array can only take on one format.	Float, Bit, UInt16, SInt16, Packed_Bit, Byte, Packed_Byte, Swapped_Byte
Data_Array_Length	Number of Data Objects. Must be larger than the data storage area required by the Map Descriptors for the data being placed in this array.	1-10,000

Example

```
//  Data Arrays

Data_Arrays
Data_Array_Name,    Data_Array_Format,    Data_Array_Length
DA_AI_01,           UInt16,               200
DA_AO_01,           UInt16,               200
DA_DI_01,           Bit,                  200
DA_DO_01,           Bit,                  200
```

4.2. Client Side Connection Descriptions

Section Title		
Connections		
Column Title	**Function**	**Legal Values**
Port	Specify which port the device is connected to the FieldServer	P1-P8[1]
Protocol	Specify protocol used	FCI_E3
Baud*	Specify baud rate Vendor documentation indicated that the only supported Baud rate was 9600. During testing we learned this was not true. The driver was tested at 57600 and 9600 baud. Testing at 115200 Baud produced intermittent result and should not be used.	**9600**, 57600 (Vendor Limitation)
Parity*	Specify parity	**None** – (Vendor Limitation)
Data_Bits*	Specify data bits	**8** (Vendor Limitation)
Stop_Bits*	Specify stop bits	**1** (Vendor Limitation)
Handshaking*	The E3 series panels do not support hand shaking.	

Example

```
// Client Side Connections

Connections
Port,          Protocol,      Baud,     Parity
P1,            FCI_E3,        9600,     None
```

[1] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

4.3. Client Side Node Descriptors

Section Title
Nodes

Column Title	Function	Legal Values
Node_Name	Provide name for Node	Up to 32 alphanumeric characters
Node_ID	Only required for networked configurations. Set the Node_ID of the local panel to zero, and create one Node Descriptor for each panel setting the Node_ID to the panel number.	Whole numbers 0,1,2,3 …
Protocol	Specify protocol used	FCI_E3
FCI_Reset_Action_Option*	Only required for networked configurations: When not specified or set to '**Reset_by_any_Node** ' then the driver will reset the data array points associated with the given node irrespective of the reset message's origin. When set to 'Reset_by_this_Node_Only' then the driver only resets the data associated with the given node if the reset originated from the same node.	**Reset_by_any_Node**, Reset_by_this_Node_Only
Connection	Specify which port the device is connected to the FieldServer	P1-P8[2]

Example

```
//  Client Side Nodes

Nodes
Node_Name,          Protocol,       Connection
Panel-01,           FCI_E3,         P8
```

4.4. Client Side Map Descriptors

4.4.1. FieldServer Related Map Descriptor Parameters

Column Title	Function	Legal Values
Map_Descriptor_Name	Name of this Map Descriptor	Up to 32 alphanumeric characters
Data_Array_Name	Name of Data Array where data is to be stored in the FieldServer	One of the Data Array names from "Data Array" section above
Data_Array_Offset	Starting location in Data Array	0 to maximum specified in "Data Array" section above
Function	Function of Client Map Descriptor.	Server, Passive_Client

[2] Not all ports shown are necessarily supported by the hardware. Consult the appropriate Instruction manual for details of the ports available on specific hardware.

4.4.2. Driver Related Map Descriptor Parameters

Column Title	Function	Legal Values
Node_Name	Name of Node to fetch data from	One of the Node Names specified in Section 4.3
Event Type	This driver uses this parameter to determine the suitability of a Map Descriptor when looking for a location to store data from an incoming message. A Map Descriptor may be defined to store only 'Alarm', 'Fault', 'Trouble' or 'Other events. Alternatively, specify "Any" A table of events vs. categories is provided in Appendix A.1	Any, Other, Fault, Alarm, Trouble
Point Type	This driver uses this parameter to determine the suitability of a Map Descriptor when looking for a location to store data from an incoming message.	Zone, Relay, Loop, Sensor, Module, Panel
Relay/Loop/Zone Number	Ignored when the Point Type is 'Panel' Point Type = Relay 1..255 Point Type = Zone 1..8 Point Type = Loop 1..2 Point Type = Module 1..2 Point Type = Sensor 1..2	Whole Numbers 1, 2 , etc
Length*	Each Map Descriptor defines storage locations for a series of addresses. This parameter specifies the length of the series.	**1**,2,3 .etc Whole numbers
Address	This parameter is specific to Map Descriptors with 'Event Type' Module or Sensor. It specifies the starting Module or Sensor number. The length parameter then determines the range of the Sensor/Module numbers	1..99
Store As*	By default the driver stores using the 'Index Value' Method. If set to 'Bit' the driver will use the primary Data Array to store using the 'Bit Storage' Method. These methods are described in Appendix A.3	Bit, **Index_Value**
DA_Bit_Name	If the default 'Store As' is specified or if the parameter is omitted then a secondary array can be specified using this parameter - the driver will store event data as 'Bit Storage' in the Secondary array (and as 'Index Values' in the Primary array.) These methods are described in Appendix A.3	One of the Data Array names from "Data Array" section above
Clear on Reset*	If this parameter is specified it will prevent the Driver resetting the Data Array points associated with the Map Descriptor on reset.	**Yes**, No

4.5. Map Descriptor Example 1 – Sensor / Module Events

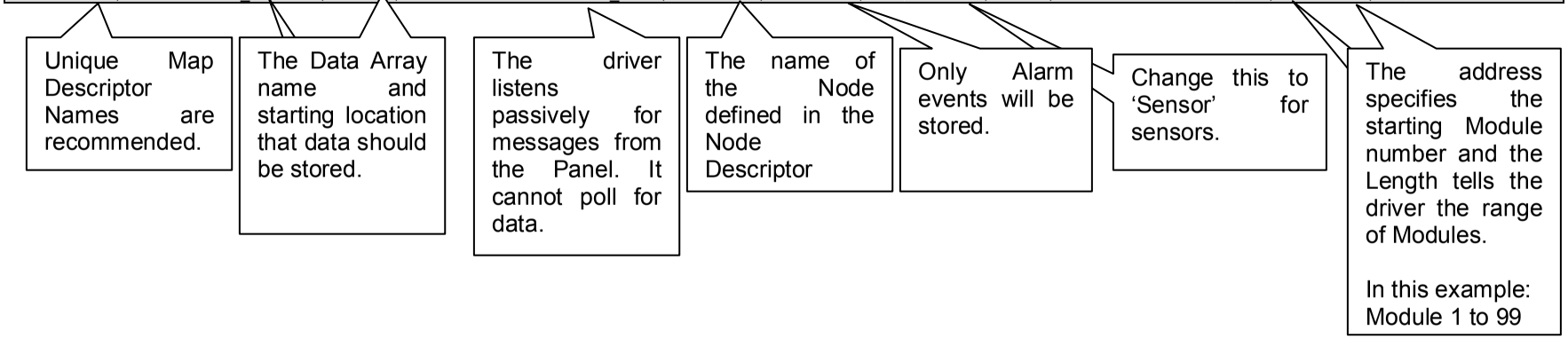
If messages from Loop 1, Module 1 to 99 are received then the Map Descriptor in this example will be used for storage. If modules occur on more than one loop, one Map Descriptor is required per loop. In this example the event type is set to 'Alarm' so only "Alarm" events will be stored. The "Event_Type" can be changed to suit the required data. If all events are required, the parameter should be set to "Any".

Example:

F.S.E.C. :[CR][LF]
FIRST ALARM: UP STAIRS N. ENT Manual Station L1M21 00:37:28 01/01/99[CR][LF]

```
//  Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name,    Data_Array_Name,    Data_Array_Offset,    Function,          Node_Name,    Event Type,    Point Type,    Relay/Loop/Zone Number,    Address,    Length,    Clear_on_Reset
ModuleData1,            DA_MODULE,          0,                    Passive_Client,    Panel-01,     Alarm,         Module,        1,                         1,          99,        Yes
```



4.6. Map Descriptor Example 2 – Bit Storage

This example defines storage location for Relay Point events. The example would work for all other point types. In the example, both primary and secondary storage Data Arrays have been specified. The driver stores index values in the primary array. Each new event for a particular relay will overwrite the value stored previously. In the Bit Array, the driver sets the bit corresponding to the event, leaving other bits unchanged – thus the secondary storage can be used to determine if more than one event is active at a time.

```
// Client Side Map Descriptors

Map_Descriptors
Map_Descriptor_Name,  Data_Array_Name,  Data_Array_Offset,  DA_Bit_Name,  Function,  Node_Name,  Event Type,  Point Type,  Relay/Loop/Zone Number,  Address,  Length,  Clear_on_Reset
RelayData,  DA_RELAY,  0,  DB_Relay,  Passive_Client,  Panel-01,  Any,  Relay,  1,  -,  4,  Yes
```

Data_Array_Name is where the primary DA is specified. Index values are stored here

DA_Bit_Name is where secondary storage is defined. Events are stored by setting appropriate bits.

Remember that 2 elements per Relay, Module, Sensor, Loop are used.

Map Descriptors for storing Relay, Loop, Zone and Panel do not need the address specified.

5. Configuring the FieldServer as a Gamewell FCI-E3 Series Serial Driver Server

The server side of the driver is intended to support FieldServer's Quality Assurance program and is not intended to provide complete emulation of an E3 and is thus not fully documented. Should you require the Server side functionality to be documented and enhanced, please contact FieldServer's sales group.

Appendix A. Advanced Topics

Appendix A.1. Events and Event Categories

The driver reports the event cause using the matching index value. There are 4 event categories:

1 = Other
2 = Fault
3 = Alarm
4 = Trouble

The message category must match the 'Event Type' parameter specified on a Map Descriptor before that Map Descriptor can be considered for storage of the message data.

Index	Category	Event
1	FCI_EVENT_TYPE_FAULT	"Fault"
2	FCI_EVENT_TYPE_OTHER	"Short"
3	FCI_EVENT_TYPE_OTHER	"Disconnect"[3]
4	FCI_EVENT_TYPE_OTHER	"Comm Fault"
5	FCI_EVENT_TYPE_OTHER	"Config Err"
6	FCI_EVENT_TYPE_OTHER	"Eeprom Bad"
7	FCI_EVENT_TYPE_OTHER	"Reset"
8	FCI_EVENT_TYPE_OTHER	"Silence"
9	FCI_EVENT_TYPE_OTHER	"Cross Zone"[3]
10	FCI_EVENT_TYPE_OTHER	"Acknwldgd"
11	FCI_EVENT_TYPE_OTHER	"Walk Test"
12	FCI_EVENT_TYPE_OTHER	"Alarm Test"
13	FCI_EVENT_TYPE_OTHER	"SPVSN Test"
14	FCI_EVENT_TYPE_OTHER	"Fault Test"
15	FCI_EVENT_TYPE_OTHER	"Fire Drill"
16	FCI_EVENT_TYPE_OTHER	"Batt Test"
17	FCI_EVENT_TYPE_OTHER	"PRGM Mode"
18	FCI_EVENT_TYPE_OTHER	"Action"
19	FCI_EVENT_TYPE_OTHER	"Loop Break"
20	FCI_EVENT_TYPE_ALARM	"Alarm"
21	FCI_EVENT_TYPE_OTHER	"P.A.S."
22	FCI_EVENT_TYPE_OTHER	"Off-Normal"
23	FCI_EVENT_TYPE_OTHER	"RZA Fault"[3]
24	FCI_EVENT_TYPE_OTHER	"Verify"[3]
25	FCI_EVENT_TYPE_OTHER	"CM Short"[3]
26	FCI_EVENT_TYPE_OTHER	"Test Fail"
27	FCI_EVENT_TYPE_OTHER	"Alert"
28	FCI_EVENT_TYPE_OTHER	"Dirty"
29	FCI_EVENT_TYPE_OTHER	"Very Dirty"
30	FCI_EVENT_TYPE_OTHER	"Missing"
31	FCI_EVENT_TYPE_OTHER	"Wrong Type"
32	FCI_EVENT_TYPE_OTHER	"Extra Addr"
33	FCI_EVENT_TYPE_OTHER	"Clock Err"[3]
34	FCI_EVENT_TYPE_TRBLE	"Trouble"[3]
35	FCI_EVENT_TYPE_OTHER	"MLT Events"[3]
36	FCI_EVENT_TYPE_OTHER	"Alrm Ackd"
37	FCI_EVENT_TYPE_OTHER	"Outpt Fail"
38	FCI_EVENT_TYPE_OTHER	"Tally Flt"
39	FCI_EVENT_TYPE_OTHER	"AC Flt To"
40	FCI_EVENT_TYPE_OTHER	"Trbl Ackd"
41	FCI_EVENT_TYPE_OTHER	"Access"
42	FCI_EVENT_TYPE_OTHER	"Netwrk Flt"
43	FCI_EVENT_TYPE_OTHER	"NetGndFlt"
44	FCI_EVENT_TYPE_OTHER	"Dact Fault"
45	FCI_EVENT_TYPE_OTHER	"Node Msng"
46	FCI_EVENT_TYPE_OTHER	"Node Xtra"
47	FCI_EVENT_TYPE_OTHER	"Fans Off"
48	FCI_EVENT_TYPE_OTHER	"Xzone Alrm"

[3] Not defined in the spec

Appendix A.2. Extending the Event Table

New event causes may be added to the Event Table and the index value or category of existing event causes modified by adding a section to the configuration CSV file. The examples below illustrate this:

Example 1: Index value of 'Trouble' is updated to a new value of 100

Driver_Table			
Event_Type_Description,	Event_Type_Index_Value,	Event_Type_Category ,	Protocol
TROUBLE,	100,	4,	FCI_E3

Example 2: New Entry is added

Since it has been added as category=3, only Map Descriptors with 'Event Type' set to “Alarm” or “Any” will capture messages with this event description

Driver_Table			
Event_Type_Description,	Event_Type_Index_Value,	Event_Type_Category,	Protocol
DESTROYED,	51,	3,	FCI_E3

For categories use the following values
'Other' = 1
'Fault' = 2
'Alarm' = 3
'Trouble' = 4

Appendix A.3. How Data is stored

All messages shorter than 102 characters are discarded unless they contain information identifying the networked/local panel. All other messages are processed as follows:

- The driver determines if the message is a Zone, Relay, Loop, Sensor, Module or Panel message.
- The driver finds all Map Descriptors with matching ‘Point Type’ parameters.
- The event category is determined.
- Map Descriptor selection is refined based on whether the ‘Event Type’ matches or has been defined as “Any”
- The driver determines the Loop. Relay, Zone, Sensor and Module numbers from the message and refines its selection of Map Descriptors by selecting those that match the values determined from the message.
- The selected Map Descriptors are now used to determine a Data Array and offset at which to store the data.
- Finally the driver checks the ‘Store As’ parameter. If it hasn’t been specified then ‘Index Value’ storage is assumed. If it has been specified as ‘Bits’ then the driver will perform ‘Bit Storage’. In cases where the Map Descriptor has both a primary and secondary Data Array, the driver will use ‘Index Value’ storage using the primary data array and ‘Bit Storage’ using the secondary array.

Example:

The following fragment is part of a Map Descriptor definition; some parameters have been omitted for the purposes of clarity.

```
Map_Descriptors,
Data_Array_Name,  Data_Array_Offset,  Event Type,  Point Type,  Relay/Loop/Zone Number,  Address,  Length,  Clear_on_Reset,  DA_Bit_Name
DA_MODU      ,    0,                  ANY,         Module,      1,                       1,        99,      Yes,             DB_MODU
DA_MODU_A,        0,                  ALARM,       Module,      1,                       1,        99,      Yes,             DB_MODU_A
DA_MODU_F,        0,                  FAULT,       Module,      1,                       1,        99,      Yes,             DB_MODU_F
DA_MODU_T,        0,                  TROUBLE,     Module,      1,                       1,        99,      Yes,             DB_MODU_T
DA_MPODU_O,       0,                  OTHER,       Module,      1,                       1,        99,      Yes,             DB_MODU_O
```

Message = “FAULT: AC Power E3 0:00:04 1/01/92”

- This message does not report the status of a Zone, Relay, Loop, Sensor or Module and is therefore assumed to be a panel message. Since there is no Map Descriptor with “Point Type” Panel, the message is ignored.

Message = “TROUBLE: QZUb L1M22 << Chief's Office >> 5:24:00 3/03/93”

- This message reports status for Loop 1 Module 22. Since all the MD’s in the example have a ‘Point Type’=‘Module’, they are all considered for storage.
- The driver looks in the Event Table and finds it has an index value of 34 and a category of 4 (Trouble). Only the MD’s with “Event Type” set to “Any” and “Trouble” are now considered.
- Since the value of the ‘Relay/Loop/Zone’ parameter matches the Loop number in the message, these MD’s remain in contention.
- The Module number of 22 is compared with the MD’s Address and Length Parameters. The Address is the starting number and the length defines the range. Both Map Descriptors have addresses of 1 and length of 99 and thus both are still selected because the Module of 22 falls in this range.
- The driver calculates an offset based on the offset specified in the Map Descriptor and the Module number relative to the Map Descriptor address:

  Map Descriptor Offset = 0
  Map Descriptor Address = 1
  Message Module = 22

  Module 1’s data is stored at offset 0 and hence Module 22’s data will be stored at offset 21. The driver stores the value 34 at offset 21 overwriting any data previously stored at that location. This is ‘Index Value’ Storage.
- Secondary storage has been defined using the ‘DA_Bit_Name’ Data Array. The driver doubles the offset as two locations are used for each address. The driver then reads the value found in the Data_Array, modifies it and writes it back. Since the index value is 34 the driver modifies the 34^{th} bit – or expressed another way, the driver modifies the 2^{nd} bit (34-32) at offset+1.
- Thus, the driver calculates the offset for Bit Storage as 2 x 21 = 42. The driver sees that bit 34 is 2 2^{nd} bit in the next offset and so the driver reads DB_MODU:43, modifies the value by setting the 2nd bit on and then writing the modified value back. During the modification all other bits are left intact. Thus using the Bit Storage method, a single Module (or sensor…) can keep track of multiple events.

Appendix A.4. Panel Synchronization

Manual synchronization is required. When the "Reset" button on the panel is pressed a message is transmitted to the FieldServer, which clears the data in the FieldServer. After a reset the panel sends messages to report all abnormal states. When all these messages have been processed the FieldServer and panel will be synchronized. This process can be repeated at any time.

Appendix A.5. What happens when the panel sends a Reset Message

When a panel sends a reset message the driver processes every single Map Descriptor, looking at the 'Clear on Reset' parameter (See section 4.4.2). If the parameter is set to yes, then the driver sets all the Data Array elements referenced by the Map Descriptor to zero by looking up the Data Array Name, the Data Array offset and the length. The driver also clears the relevant sections of a Data Array specified with the DA_Bit_Name parameter.

The process can take some time. For this reason, it is suggested that you take care not to set the Map Descriptor length to a value larger than necessary.

Appendix A.5.1. Networked Panels.

The driver can processes messages and store data from multiple panels provided that:

- The panels are connected in an FCI network and the panels are configured to report their events to the main panel
- The main panel is configured to send the Node of origin in a message preceding the event message. Consult with FCI for information on how to achieve this.

Example of message sent by a panel that is networked. The driver is dependent on seeing the node of event origin included in parenthesis before each event message.

```
Node02:
MISSING                                  Acclimate       L1S041 00:17:06
01/01/06

Node55: 1st Floor Lobby
DISCONNECT RSTRD                         Ion Detector    L1S024 09:51:44
01/15/06

Local:
FIRST ALARM                              Photo Detector  L1S001 09:24:52
12/01/06
```

The main panel is identified as 'Local'. The driver interprets this as Node_ID=0.

To capture events from multiple networked panels a Node Descriptor is required for each panel with the appropriate Node_ID. Each Node requires a set of Map Descriptors.

Appendix B. Driver Error Messages.

Message	Description
FCI_E3:#1 FYI. Use a DA called <%s> to expose diagnostic info., FCI_E3_STATS_DA)	Refer to Appendix B.1
FCI_E3:#2 FYI. Added Event Desc=<%s> Index=%d Categ=%d , new_event_desc , new_event_desc_index_value , new_event_desc_categ) ;	Printed for information only. No action required if it confirms your expectations.
FCI_E3:#3 Err. No space. Reject Event Desc=<%s> Index=%d , new_event_desc , new_event_desc_index_value) ;	There is only space for 60 event types.[4]
FCI_E3:#4 FYI. Duplicate Event Desc=<%s>. , new_event_desc) ;	The Event type being added already exists. If you are updating the category, ignore the message. Otherwise correct the configuration file.[4]
FCI_E3:#5 FYI. Duplicate Event Desc=<%s>. , new_event_desc) ;	
FCI_E3:#6 Err. Event Index=%d. Too big to set bit., drv_bd->event_index)	If the event index is greater than 64 then the data cannot be stored as bits - only 64 bits are reserved for events.
FCI_E3:#7a Err. DA=%s too short. Rqd=%d, dt->buffer_name , offset);	The MD in question has a length and offset which makes it run past the end of the Data Array. Message 7b is printed when data is being stored as bits.[4]
FCI_E3:#7b Err. DA=%s too short. Rqd=%d, possible_md->data->buffer_name , offset);	
FCI_E3:#8 FYI. Reset was rcvd and processed! Stamped %s %s , drv_bd->time , drv_bd->date)	Printed for information only. No action required.
FCI_E3:#9 Err. Reset was ignored.	A reset was received but the driver could not reset any data. 'Clear_on_Reset' possibly set to 'no' on all Map Descriptors?
FCI_E3:#10 FYI. Reset of DA=%s Off=%d Len=%d, possible_md->data->buffer_name , possible_md->bxi_data_buffer_offset , possible_md->data_length);	Printed for information only. No action required.
FCI_E3:#11 Err. Cant reset DA=%s len=%d rqd=%d, possible_md->data->buffer_name , da_get_length_in_items ((DAH_TYP) possible_md->data) , possible_md->data_length+possible_md->bxi_data_buffer_offset);	The Map Descriptor in question has a length and offset which makes it run past the end of the Data Array.[4]
FCI_E3:#12a Err. No MD's to store message data.	The Driver could not find a place to store data from a message received. If the data is not required then the message may be ignored. Otherwise update the configuration file[4].
FCI_E3:#12b Err. No MD's to store message data."	
FCI_E3:#13 Err. Msg was ignored. MD Required for Storage.	
FCI_E3:#13a Err. Diagnostic 1);	Take a log. Try and repeat the event that caused the message to be printed. Then contact tech support.
FCI_E3:#13b Err. Diagnostic 2);	
FCI_E3:#13c Err. Diagnostic 3);	
FCI_E3:#14 Err. <%s> file not found., md->mapdesc_name) ;	If this error is repeated often it is possible that a FCI firmware update has made the driver unusable. Take a log and contact tech support.
FCI_E3:#15 Err. Event Type=<%s> Not recognized." , drv_bd->event_desc)	
FCI_E3:#16 Err. Point Type='%c'(%#x) Not recognized. , drv_bd->point_identifier[0] , drv_bd->point_identifier[0]) ;	

[4] Correct the configuration file, download to the FieldServer and restart the FieldServer for the changes to take effect.

Message	Description
FCI_E3:#17 Err. Loop=%d < 1. Rejected. , drv_bd->loop)	This message should only be printed if a byte in a message has been corrupted. If you notice it more than once then take a log and contact tech support.
FCI_E3:#18 Err. Loop Type='%c'(%#x) Not recognized. , drv_bd->point_identifier[2] , drv_bd->point_identifier[2])	If this error is repeated often it is possible that a FCI firmware update has made the driver unusable. Take a log and contact tech support.
FCI_E3:#19 Err. Relay=%d < 1. Rejected. , drv_bd->relay	This message should only be printed if a byte in a message has been corrupted. If you notice it more than once then take a log and contact tech support.
FCI_E3:#20 Err. Zone=%d < 1. Rejected. , drv_bd->zone"	
FCI_E3:#21 Err. Point Type not recognized	Valid Point Types are listed in section 4.4.2[5]
FCI_E3:#22 Err. Undefined Point Type"	
FCI_E3:#23 Err. Event Type not recognized	Valid Event Types are listed in section 4.4.2[5]
FCI_E3:#24 Err. Undefined Event Type	
FCI_E3:#25a Err. Address+Length>99. Length Truncated	The maximum value for a sensor/module is 99. The combination of address and length specified produce a number > 99[5]
FCI_E3:#25b Err. Address+Length>99. Length Truncated	
FCI_E3:#26 Err. Invalid Module number. Expected 1..99	Correct the configuration file[5]
FCI_E3:#27a Err. Invalid Loop number. Expected 1..10	
FCI_E3:#27b Err. Invalid Loop number. Expected 1..10	
FCI_E3:#27c Err. Invalid Loop number. Expected 1..10	
FCI_E3:#28 Err. Invalid Sensor number. Expected 1..99	
FCI_E3:#29 Err. Invalid Zone number. Expected 1..255	
FCI_E3:#30 Err. Invalid Relay number. Expected 1..255	
FCI_E3:#31 Err. Point Type Invalid.	Valid Point Types are listed in Section 4.4.2[5]
FCI_E3:#32 Err. No MD Length. Default to 1	The length of each Map Descriptor should be specified. Refer to Section 4.4.2[5]
FCI_E3:#33 Err. Driver can't poll or write.	The driver can only listen passively for message from the panel. Map Descriptors may not be configured as active. [5]
FCI_E3:#36 Err. Too Short. Bytes=%d , conn->ux_iptr"	An event message is less than 80 bytes long. If this error is repeated often it is possible that a FCI firmware update has made the driver unusable. Take a log and contact tech support.

[5] Correct the configuration file, download to the FieldServer and restart the FieldServer for the changes to take effect.

Appendix B.1. Driver Stats Exposed.

In addition to the standard FieldServer operating statistics the driver exposes certain key stats in a Data Array if required. An upstream device can then monitor these stats.

Add the following to your configuration file to activate these stats.

```
// Expose Driver Operating Stats.

Data_Arrays
Data_Array_Name,        Data_Format,    Data_Array_Length
fci-e3-stats,           UINT32,         200
```

The driver exposes stats based on a port handle. The offset specified in the table below must be added to the handle number multiplied by 100. i.e. for port whose handle is 1 then the driver will store the 1st stat at 1+100*1=101.

Stat	Offset	Description
#define FCI_E3_STAT_NO_PLACE_TO_STORE	1	Increments each time point data is received but there is no Map Descriptor to store the data (any)
#define FCI_E3_STAT_NO_PLACE_TO_STORE_ZONE	2	Increments each time point data is received but there is no Map Descriptor to store Zone data
#define FCI_E3_STAT_NO_PLACE_TO_STORE_RELAY	3	Increments each time point data is received but there is no Map Descriptor to store Relay data
#define FCI_E3_STAT_NO_PLACE_TO_STORE_LOOP	4	Increments each time point data is received but there is no Map Descriptor to store the Loop data
#define FCI_E3_STAT_NO_PLACE_TO_STORE_SENSOR	5	Increments each time point data is received but there is no Map Descriptor to store the Sensor data
#define FCI_E3_STAT_NO_PLACE_TO_STORE_MODULE	6	Increments each time point data is received but there is no Map Descriptor to store the Module data
#define FCI_E3_STAT_EMPTY_MSG	7	Number of times that a message line was zero bytes long (excluding the terminator)
#define FCI_E3_STAT_SHORT_MSG	8	Number of times that a message line was too short probably a system id tag line
#define FCI_E3_STAT_NO_RESET	9	Increments each time a reset was rcvd but no Data Array was reset
#define FCI_E3_STAT_NO_PLACE_TO_STORE_PANEL	10	Increments each time point data is received but there is no Map Descriptor to store data that cannot be attributed to a zone, relay, loop, sensor, module
#define FCI_E3_STAT_RCVD_MSGS	11	Increments each time a message is received

Stat	Offset	Description
#define FCI_E3_STAT_RCVD_BYTES	12	Increments each time a character is received from the panel. The bytes are only added when a message terminator is received. Thus this count is equivalent to the byte count in all FCI_E3_STAT_RCVD_MSGS
#define FCI_E3_STAT_PARSED_NO_ERRS_EXCLD_RESET	13	Increments each time a message is parsed without errors. Excludes Reset Messages
#define FCI_E3_STAT_PARSED_NO_ERRS_RESET	14	Increments each time a reset message is parsed without errors.
#define FCI_E3_STAT_PARSED_NO_ACTION	15	Increments each time a message is parsed with no errors but the nature of the message doesn't require data to be stored. (eg empty message lines)
#define FCI_E3_STAT_PARSED_WITH_ERRS	16	Increments each time a message produces an error when parsed.
#define FCI_E3_STAT_INHIBIT_RESET	17	Set to 1 to inhibit resets altogether
#define FCI_E3_STAT_INHIBIT_RESET_DA_PUT	18	Set to 1 to inhibit resets from clearing arrays
#define FCI_E3_STAT_INHIBIT_RESET_WHILE	19	Set to 1 to inhibit the reset function from looping through Map Descriptors
#define FCI_E3_STAT_NODE_INFO_MSG	20	Increments each time a message with Node information is received
#define FCI_E3_STAT_NO_PLACE_TO_STORE_NODE	21	Increments each time an event message containing Node information relating to a Node that cannot be found is discarded.